Visual Comput (2006) 22: 631–641
DOI 10.1007/s00371-006-0053-z

ORIGINAL ARTICLE

Michael Waschbüsch
Stephan Würmlin
Markus Gross

Interactive 3D video editing

Published online: 15 August 2006

M. Waschbüsch (✉) · S. Würmlin · M. Gross
Computer Graphics Laboratory, ETH Zürich, Switzerland
{waschbuesch, wuermlin, grossm}@inf.ethz.ch

Abstract We present a generic and versatile framework for interactive editing of 3D video footage. Our framework combines the advantages of conventional 2D video editing with the power of more advanced, depth-enhanced 3D video streams. Our editor takes 3D video as input and writes both 2D or 3D video formats as output. Its underlying core data structure is a novel 4D spatio-temporal representation which we call the *video hypervolume*. Conceptually, the processing loop comprises three fundamental operators: *slicing*, *selection*, and *editing*. The slicing operator allows users to visualize arbitrary hyperslices from the 4D data set. The selection operator labels subsets of the footage for spatio-temporal editing. This operator includes a 4D graph-cut based algorithm for object selection. The actual editing operators include cut & paste, affine transformations, and compositing with other media, such as images and 2D video. For high-quality rendering, we employ EWA splatting with view-dependent texturing and boundary matting. We demonstrate the applicability of our methods to post-production of 3D video.

Keywords 3D video · Video editing · Video processing · Point-based graphics · Graph cuts

1 Introduction

In recent years, significant progress has been made in the acquisition of dynamic three-dimensional objects and scenes. Besides numerous prototype systems for depth image scanning, commercial sensors, such as 3DV Systems' ZCam™(http://www.3dvsystems.com), have become available and provide solutions for studio setups. The major application of such systems is to facilitate foreground–background separation and other operations for post-production of conventional 2D video footage. While this technology still offers room for improvement, we can expect reliable, depth-enhanced video acquisition in the years to come. At the same time, three-dimensional and view-independent video has emerged as a novel media technology enabling a variety of 3D special effects. In most cases, multi-view video streams are combined into a spatio-temporal representation which can be re-rendered using view interpolation. It has turned out that adding geometry greatly helps to achieve production quality when interpolating between views of sparsely sampled cameras. Some approaches compute depth implicitly from the 2D video data [30] using vision algorithms, whereas others [28] explicitly assume geometry or adapt template geometry [5]. In spite of the aforementioned activities,

however, relatively little research effort has been devoted to the problem of efficient and intuitive editing of three-dimensional video.

In this paper, we present a generic framework for interactive editing of 3D video footage. It extends on existing concepts for two-dimensional video and combines their conceptual simplicity with the power of depth-enhanced video data. The multi-dimensional, spatio-temporal nature of 3D video leaves its editing highly non-trivial, but, at the same time, allows for a variety of novel features. Our framework is based on explicit 3D geometry and assumes its availability through some 3D acquisition system. Its design involves various novel features and methods leading to the following main contributions: First, we developed a novel 4D spatio-temporal representation – the so-called *video hypervolume* – for intuitive handling and editing of the four-dimensional data. The video hypervolume irregularly samples the four-dimensional space-time domain to represent dynamic 3D scenes. Second, we designed a concept for video editing which is based on three fundamental operators: *slicing*, *selection*, and *editing*. In particular, we present a 4D object selection algorithm based on graph-cuts. To convey object boundaries the user indicates object and non-object regions in the spatio-temporal domain by painting on the surfaces with a 3D paintbrush. In addition, we provide a set of spatio-temporal editing operations, such as cut & paste and affine transformations. By using the operators the processing of 3D video becomes easy and intuitive.

1.1 Related work

3D video. Over the last years, three-dimensional video emerged as a novel media technology for re-rendering of multi-view video data. It can be seen as a natural extension of 2D video to the spatio-temporal domain. On the one hand, 2.5 D representations as used in the image-based visual hulls approach (IBVH) [11] extend video imagery with view-dependent depth of the acquired surfaces as obtained by computer vision methods. While the IBVH employs shape-from-silhouettes methods, Zitnick et al. [30] use depth-from-stereo algorithms for this purpose. The latter renders novel imagery using view interpolation. Simple editing tasks can be performed, such as cloning and time-shifting of objects. However, due to its view-dependent nature complex editing tasks are not easily possible. On the other hand, there exist systems with explicit 3D geometry, such as triangular meshes [5, 17] or 3D point samples [24, 28, 29], featuring non-uniform, view-independent handling of captured objects or scenes. However, none of these approaches tackled the challenge of designing an editing system for 3D video. One common editing task for seamless cut & paste of objects into new environments would be relighting. However, as shown by Theobalt et al. [21] extracting reflectance properties from time-varying multi-view video data is a challenging task and no solution for high-quality re-rendering has been proposed so far.

Image and video editing. Our work is inspired by 2D video editing where a variety of visual effects production tools exist to support typical 2D video editing tasks. But this process is often tedious, time-consuming and sometimes only possible by introducing constraints on the scene. For instance, extracting foreground objects from the background is only possible for objects recorded in front of a special-coated background. However, without the use of dedicated backgrounds, i.e., in natural environments, real-time keying becomes much more difficult. Yet, recent research targets video cutout with arbitrary backgrounds using manual interaction and graph-cut techniques [9, 23] by extending image cutout methods [10, 18]. Our 3D video selection framework is inspired by these approaches. The video cube [7, 8] – also employed by Wang et al. for their video object cutout – is based on the concept of displaying video data as a three-dimensional volume where arbitrary slices through the volume generate spatio-temporal visualizations. While similar to our video hypervolume representation the additional dimension and irregularity of 3D video data introduces some more challenges. Proscenium [2] is a video editing framework which uses and extends the video cube representation by distortion and warping operators.

Geometry information can also be used to facilitate conventional 2D video editing. Snavely et al. [20] utilize depth maps to stylize movies with various non-photorealistic rendering techniques.

1.2 Overview

Our system complements the 3D video acquisition and reconstruction pipeline with an editing framework for post-production as illustrated in Fig. 1. It is based on a novel four-dimensional data model which represents appearance

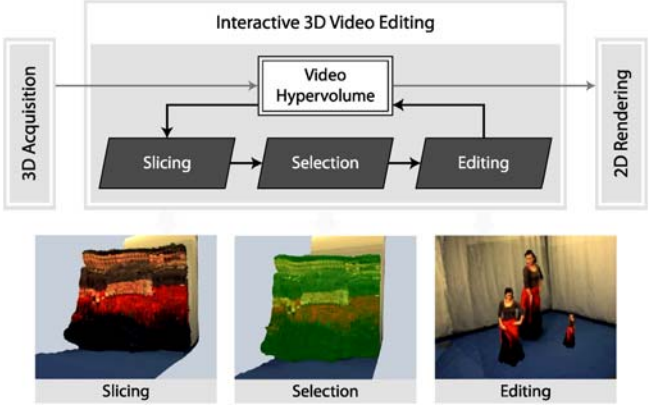

Fig. 1. The 3D video editing framework

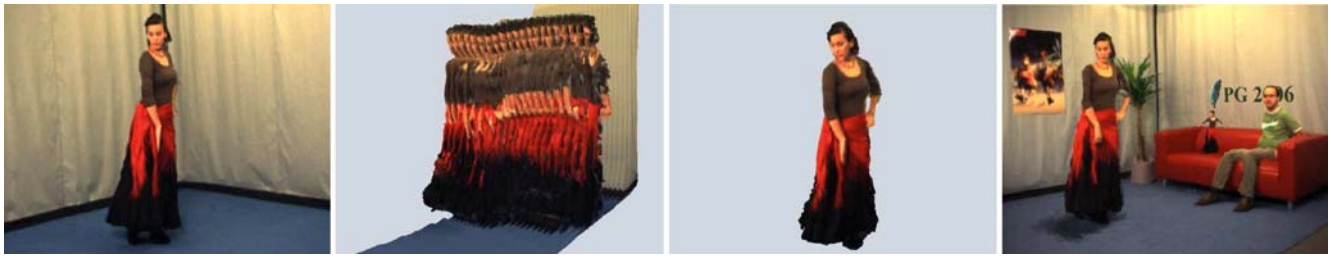

Fig. 2. Our interactive 3D video editing combines the advantages of 2D video editing with depth-enhanced 3D video streams. From left to right: Interpolated view of the 3D video input data; hyperslicing to reveal the time domain where selection and editing is intuitive and easily performed; cutout of a 3D video object; composite 3D video with additional 2D and 3D objects, new background and shadow mapping

and geometry of the scene as point-samples in spacetime. We call this representation a video hypervolume (Sect. 2). Some implementation issues are discussed in Sect. 6.

The editing framework is based on three operators: slicing, selection and editing. The *slicing* operator (Sect. 3) provides an intuitive interface to interact with the four-dimensional domain. It transforms selected parts of the 4D data set from the video hypervolume to a cloud of 3D point samples. The slice orientation and position can be changed interactively. With the *selection* operator (Sect. 4) the user can mark regions or objects of interest. While the former can be performed using marquee, lasso or paintbrush tools, the latter requires the notion of object boundaries which we introduce using a graph-cut selection scheme. Users guide the selection process by painting with an object brush and with a background brush. If the object is disconnected from other scene parts, object segmentation is often very easy due to the underlying 3D geometry – unlike in 2D video editing. All selected parts can be modified by a set of *editing* operators (Sect. 5). Operations make use of the explicitly modeled scene geometry and include cut & paste, spatial and temporal translations, rotations and scaling. During compositing handling of occlusions is provided for free. Our unified handling of space and time naturally supports editing operations exploiting both spatial and temporal coherence. Selection and editing are applied directly on a cloud of 3D point samples which can be rotated interactively using an arcball interface. The invisible, fourth domain can only be accessed by defining a different slice. Upon completion of an editing operation the data in the current slice is back-propagated into the video hypervolume. By operating on the slice only, we leverage interactive editing of the huge 3D video data sets. A typical editing session is illustrated in Fig. 2.

2 Video hypervolume

Interactive editing of 3D video footage requires a primitive and a data representation that allows for unified handling of space and time. We choose to build our editing framework on irregular point samples in the four-dimensional spacetime domain. Each point sample represents a point on a scene surface with a positional coordinate (x, y, z) and a time coordinate t. The point sample has some nice properties for representing 3D video data samples. Primarily, it does not need explicit topology and, hence, no connectivity information has to be constructed and maintained over time. Secondly, it encodes explicit 3D scene geometry and color in a homogeneous way. Furthermore it can be flexibly extended to contain more application-specific data like more sophisticated material properties or object labels. Inspired by research on visualization of time-varying volumetric data [1, 13], we introduce the *video hypervolume* as representation of the point-sampled data. Thereby, space and time are considered as compound entities and facilitate the design of a user interface to conveniently support editing operations, similar to video cubes for 2D video [7, 8]. By applying hyperslicing and projection methods (see Sect. 3), we can exploit both spatial and temporal coherence during editing. Figure 3 illustrates the video hypervolume.

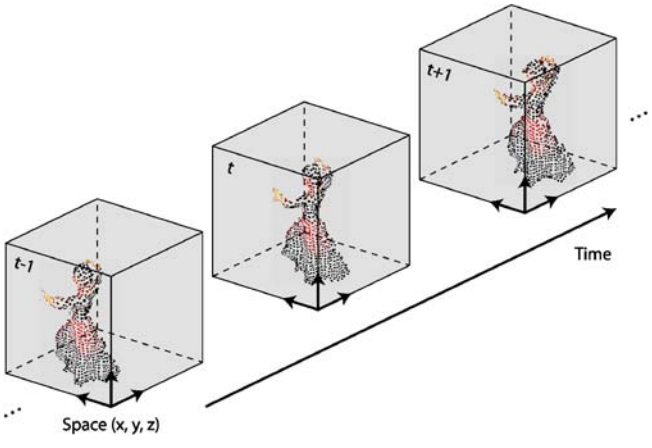

Fig. 3. The video hypervolume

The point-sampled video hypervolume fits nicely into existing frameworks for processing of point-sampled geometry [31]. This enables to utilize a variety of post-processing operations for outlier removal [25], redundancy elimination [19], and geometry smoothing [14]. Many of these algorithms are independent of the number of dimensions and, thus, can naturally be extended to integrate time coherence. For instance, we reduce redundancy in the supplied 3D video data using a point clustering algorithm [15].

Data model. The video hypervolume can be constructed independently from the 3D video acquisition system, using e.g. depth and color images as an input. It does not impose any constraints on the setup of the acquisition cameras as long as occlusions can be resolved. Each point sample in the video hypervolume carries a set of attributes describing local surface properties like position, orientation and color. Identification of a specific sample is done via its position attribute $\boldsymbol{p} = (x, y, z, t)^T$ which is a vector in Euclidean spacetime $\mathbb{R}^4$. In the spatial domain, the samples are irregularly placed on the surfaces, whereas in time we usually deal with regular sampling resulting from distinct video frames of the acquisition system. In terms of storage efficiency, the hypervolume has some advantages over a dense regular grid due to the sampling irregularity and the level of sparsity of 3D video data. Designing in-core and out-of-core compression schemes is left for future work.

The point samples in the video hypervolume can be easily constructed by back-projecting the image pixels from the acquisition cameras using the corresponding depth information. To generate hole-free renderings as output, each projection of a point onto the screen has to cover a certain area of pixels. The traditional method for static point clouds is to use surfels [16], which are small 2D ellipses tangentially aligned to the surface. In our data model, surfels would provide a full surface coverage in the spatial domain only. To also cover the time domain, we generalize them to 4D hypersurfels representing small ellipsoidal hypervolumes in $\mathbb{R}^4$. A hypersurfel is constructed from four orthogonal vectors $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_4$, $\boldsymbol{t}_i = (t_{xi}, t_{y_i}, t_{zi}, t_{ti})^T$ spanning a 4D Gaussian ellipsoid with covariance matrix $\boldsymbol{V} = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n) \cdot (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n)^T$. The first three vectors describe a conventional surfel, i.e., a 2D Gaussian ellipse embedded in the 3D spatial domain. Hence, $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are tangentially aligned to the surface with $t_{x1} = t_{x2} = 0$, and $\boldsymbol{t}_3$ is set to zero. The time domain is covered by $\boldsymbol{t}_4 = (0, 0, 0, \Delta t)^T$, where Δt denotes the temporal sampling density which corresponds to the frame rate of the video.

The tangent vectors $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ may be obtained by any of the available point-based algorithms, e.g., [31]. Alternatively, if the acquisition system is able to provide stable depth gradients, they can be computed by back-projecting the footprint of an image pixel to three-space. Based on the projection $\boldsymbol{x} = \boldsymbol{C}^{-1} \cdot d \cdot (u, v, 1)^T + \boldsymbol{c}$ of a pixel with coordinates (u, v) and depth d to a 3D point $\boldsymbol{x}$ via a camera with projection matrix $\boldsymbol{C}$ and center $\boldsymbol{c}$, the tangent vectors can be computed by differentiation as

$$(t_{x1}, t_{y_1}, t_{z1}) = \boldsymbol{C}^{-1} \cdot (d \cdot (1, 0, 0)^T + d_u \cdot (u+1, v, 1)^T), \tag{1}$$

$$(t_{x2}, t_{y_2}, t_{z2}) = \boldsymbol{C}^{-1} \cdot (d \cdot (0, 1, 0)^T + d_v \cdot (u, v+1, 1)^T), \tag{2}$$

where d_u and d_v are the directional derivatives of the depth map.

3 Slicing

To be able to view data from the video hypervolume we have to generate different views on the data set by projecting its four-dimensional data into 2D screen space. This process is called *slicing*. Apart from the standard projection showing a 3D video frame at a single time instant, arbitrary projections can be used to visualize and edit the underlying data. Slicing is performed using a two-stage projection. First, in a process called hyperslicing [27], a subset of the 4D point samples is projected to 3D. The resulting 3D point cloud is then displayed using conventional point rendering methods, see Sect. 6. Note that the actual projection operations are only carried out in the rendering process. Editing tools all operate on the original 4D points of the current hyperslice in order not to lose information. Our editing system allows one to perform hyperslicing arbitrarily, providing the user with views of both spatial and temporal scene information. This facilitates the application of editing operations in space and time. In the following, we first describe mathematical details of hyperslicing and then present the user interface for navigating in the video hypervolume.

3.1 Hyperslicing

Hyperslicing extracts a three-dimensional subspace from the 4D volume by intersection with a hyperplane. It selects all points $\boldsymbol{p} \in \mathbb{R}^4$ fulfilling the plane equation $\boldsymbol{n} \cdot \boldsymbol{p} - d = 0$, where $\boldsymbol{n} \in \mathbb{R}^4$ is the normal of the plane and d its distance from the origin. To comply with the sparse, irregular sampling of our video hypervolume, we extend this procedure as depicted in Fig. 4 and select all points within a specific distance Δd from the plane by solving

$$|\boldsymbol{n} \cdot \boldsymbol{p} - d| \leq \Delta d. \tag{3}$$

Three-dimensional positions $\boldsymbol{p}' = \boldsymbol{P}_t \cdot \boldsymbol{R} \cdot \boldsymbol{p}$ are obtained by a rotation $\boldsymbol{R}$ of $\boldsymbol{p}$ into a coordinate system locally

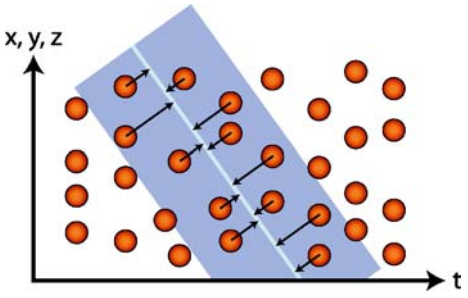

Fig. 4. Samples of the video hypervolume (orange) are intersected with the hyperslice (blue) and projected onto its center plane

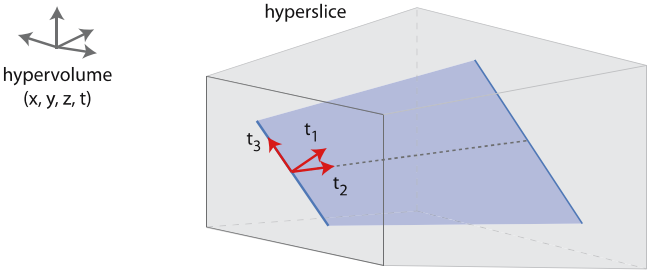

Fig. 5. Non-orthogonal slices are generated by calculating the required rotation matrix simply from a line drawn onto the screen

aligned at the hyperplane followed by a parallel projection $\boldsymbol{P}_t$ along the t-axis. The local coordinate system is spanned by four orthonormal column vectors $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_4$ with $\boldsymbol{t}_1 = \boldsymbol{n}$, yielding $\boldsymbol{R} = (\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3, \boldsymbol{t}_4)^T$. For 2D rendering of the projected 3D point samples using EWA volume splatting (see Sect. 6), the covariance matrices have to be projected accordingly by computing $\boldsymbol{V}' = \boldsymbol{P} \cdot \boldsymbol{V} \cdot \boldsymbol{P}^T$ which results in descriptions of three-dimensional Gaussian ellipsoids.

3.2 User interface

The slicing operator is used to navigate in the video hypervolume. In the most common case, the slice is orthogonal to the t axis and corresponds to a single 3D video frame. Orientation of the 3D point cloud can be controlled interactively using an arcball interface. The user can select a specific frame using a slider to control the slice position d in time. Moreover, he can adjust its thickness Δd by defining in and out points – quite similar to 2D video processing – resulting in multiple frames getting displayed. This easily allows one to identify static and dynamic scene parts.

For spatio-temporal editing it is also interesting to visualize the time domain on the screen. This facilitates intuitive spatio-temporal selection as described in the next section. The user can define arbitrarily oriented slices by drawing a line on the screen representing the hyperplane. This conveniently allows one to generate slices through a specific object of interest, as can be seen in Fig. 2b. The slider now generally controls the movement of the slice through the video hypervolume. The slice thickness can be increased such that a greater part of the orthogonal, fourth dimension gets projected onto the screen. When the user defines the new slice, the system automatically computes its rotation matrix $\boldsymbol{R}$ by determining its local coordinate system according to the drawn line and the current view. Figure 5 shows the vectors $\boldsymbol{t}_1$, $\boldsymbol{t}_2$ and $\boldsymbol{t}_3$ which are all constructed within the current hyperslice. $\boldsymbol{t}_1$ and $\boldsymbol{t}_3$ are located in the current image plane, $\boldsymbol{t}_1$ is orthogonal to the drawn line and defines the normal of the new slice. $\boldsymbol{t}_2$ is orthogonal to the image plane. $\boldsymbol{t}_4$ is not depicted as it is orthogonal to the current hyperslice.

4 Selection

The selection operator is the key to subsequent 3D video editing tasks. The 4D data does not feature object labels indicating conceptually connected data samples. For this purpose, our framework provides a graph-cut-based algorithm to associate such labels for further editing operations. But first we introduce some basic selection tools.

4.1 Selection tools

The user can view and select objects both in space and along trajectories in time using the slicing operator. By taking advantage of the underlying 3D geometry, one can often already accurately select objects and regions of interest, using basic selection tools.

Marquee and lasso selection tools. Similar to 2D photo editing applications our framework provides marquee and lasso selection tools. With these tools users are intuitively able to select large areas of the visualized slice. Users draw 2D regions on the screen which get extruded into the slice domain for 3D selection using the current virtual camera parameters. In this way a whole subvolume and possibly hidden surfaces are selected. However, by rotating the viewed data the user easily sees where hidden surfaces are selected and can work on the selection using different selection modes. Our framework provides addition, difference and intersection modes.

3D paintbrush. Another selection tool is the paintbrush also known from 2D photo and video editing applications to paint selections or colors. However, due to the additional dimension we have to define a 3D footprint of the paintbrush. Intuitively we define the 3D paintbrush

as a spherical volume in 3D. We determine the 3D center point by calculating the 3D position of the front surface data sample at the 2D screen-space coordinate of the mouse pointer. 3D data points are selected if they are contained in the spherical volume around the center point and are determined using a range query in the underlying kd-tree structure (see Sect. 6). We calculate the 3D center point with help of the z-buffer. By considering z-values of all pixels within the screen-space footprint of the projected spherical volume the intersection point of the picking ray with the scene can be determined very robustly. The user can chose the depth of the sphere as either determined by the nearest z-value or by a median filter over all z-values. The former can be used for selecting small surface patches in front of a bigger surface without the need to exactly click on them. The latter can be employed for selecting the densest surface without considering small surface patches.

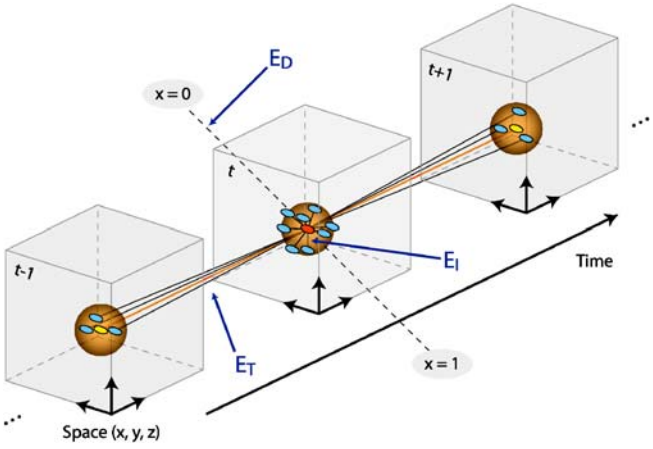

Fig. 7. When constructing the 4D graph, any data sample or data region contributes to the graph according to intra-frame and inter-frame neighborhoods and energies, as well as to virtual nodes with data energies

4.2 Object selection

The captured 4D data set does not supply the user with object boundaries or labels. However, for complex editing tasks such a labeling is essential. Therefore, we introduce a graph-cut-based algorithm to associate such labels. We use the 3D paintbrush and the marquee selection tools introduced in the previous section to specify the necessary constraints. With the paintbrush, the user can mark surface patches which should be selected (red paint) and patches which should not be selected (blue paint). The marquee selection tools (green paint) are used to define a spatio-temporal region of interest, thereby excluding large uninteresting regions and speeding up the min-cut optimization significantly. Performing the optimization on whole 3D video data streams is not feasible interactively.

The status of all data samples marked with green paint is then determined by invoking a min-cut optimization after hitting a button in the interface. Upon completion, the user can refine his markings using the selection tools and run the optimization again. Figure 6 illustrates the object selection operator.

Fig. 6. Object selection. Left: The user wants to select a person and marks her with red paint, the floor with blue paint and the region of interest with a rectangular marquee selection. Middle: The first invocation of the min-cut optimization wrongly marked samples on the wall behind the person. Right: After specifying more paintbrush strokes, the optimization completes with a satisfying selection

4.3 Graph construction

The object selection problem can be interpreted as a graph labeling problem. Each data sample is assigned a unique label $x \in \{1(S), 0(N)\}$ where 1 means the data sample belongs to the selection (S) and 0 that it does not belong to the selection (N). We construct a 4D graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{A} \rangle$ on the 4D hypervolume bounded by the region of interest. The node set $\mathcal{V}$ are all data samples which have been defined as the spatio-temporal region of interest. A node u represents a data sample p_i with label x_i, color $\boldsymbol{c}_i$, surface normal $\boldsymbol{n}_i$, as well as possibly a user assigned label γ_i. Unassigned nodes are tagged as $\emptyset$. The position of the data sample is not considered in the graph directly. However, we require the positional information to generate the graph arcs. Furthermore, we define $\mathcal{S}_t$ as the scene at a time instant t. Figure 7 illustrates the 4D graph construction.

We construct the intra-frame arcs $\mathcal{A}_I$ by connecting *spatially* adjacent data samples in the same time instant S_t. The data samples are irregularly sampled in space-time and do not feature connectivity. Hence, we have to calculate the spatially adjacent samples by using range queries – quite contrary to similar approaches in 2D video cutout [9] which have explicit neighborhoods on the pixel grid. We apply a 3D Kd-tree (see Sect. 6) for this purpose and generate arcs for all data samples which lie inside a sphere with given radius (orange sphere in Fig. 7), typically 0.02 m in our metric environment. We take the nearest k data samples and exclude evidently unrelated samples with mean color or normal differences over a certain threshold (usually 0.1).

Inter-frame arcs $\mathcal{A}_T$ connect *temporally* and *spatially* adjacent point samples in adjacent time instants $S_{t\pm1}$ that are located within a given 3D radius, typically 0.04 m. We also use a Kd-tree for this purpose, created in the corresponding time instants $t \pm 1$. We initialize the range query by projecting the data sample (orange point in Fig. 7) from time t to $t \pm 1$ (yellow points in Fig. 7). Note that the radius has to be higher than for the intra-frame arcs due to non-regular sampling and motion. Furthermore, we only take the $k/2$ nearest data samples and exclude unrelated data samples too. In our current implementation we set k to 8.

4.4 4D graph-cut optimization

Similar to work on 2D video cutout [9, 23] we define a cost function E on the constructed graph $\mathcal{G}$. The 4D graph-cut algorithm then solves the object labeling problem by minimizing the following energy:

$$E(P, \Gamma) = \sum_{u \in \mathcal{V}} E_D(p_i, \gamma_i) + \lambda_I \sum_{(u,v) \in \mathcal{A}_I} E_I(p_i, p_j) + \lambda_T \sum_{(u,v) \in \mathcal{A}_T} E_T(p_i, p_j) \quad (4)$$

The optimization assigns labels x_i for each data sample p_i in node i. P denotes the solution to this problem with user assigned labels Γ. Figure 7 illustrates the different terms of the energy function. E_D is the likelihood energy while E_I and E_T are the prior energies. E_D measures the accordance of the color of a data sample to the color models assembled from the user-assigned labels. E_I and E_T assess the color and geometry differences between spatially and temporally adjacent samples. They penalize strong color and normal deviations and ensure spatial and temporal coherence in the selection process. We employ the max-flow algorithm from [3] to minimize the energy $E(P)$ in Eq. 4.

Likelihood energy E_D. We assemble two Gaussian mixture models (GMM) by sampling the color $\boldsymbol{c}_i$ of the data samples with user-assigned labels [18]. One GMM is built for the "selected" samples $\gamma_i = S$ and one for the samples $\gamma_i = N$ marked as not to be selected. The likelihood energy can then be defined as:

	$\gamma_i = S$	$\gamma_i = N$	$\gamma_i = \emptyset$
$E_D(p_i = S)$	0	∞	$\frac{D_S}{D_S + D_N}$
$E_D(p_i = N)$	∞	0	$\frac{D_N}{D_N + D_N}$

The energies in the first two columns ensure that the user-assigned labels are not violated with the optimization procedure. Since most nodes have no label ($\gamma_i = \emptyset$), we have to calculate data costs for these nodes D_S and D_N. As in [18] they are normalized for determining the final energy (third column). To check whether the data samples belong to the selected or unselected region we determine the negative log-likelihood D_S and D_N using the GMMs:

$$D_{S|N}(p_i = S \mid N) = -\log \sum_{k=1}^{K} w_{k,S|N} e^{-\frac{1}{2}(c_i - \mu_{k,S|N})^T \Sigma_{k,S|N}^{-1} (c_i - \mu_{k,S|N})} \quad (5)$$

μ_k and $\sum_k$ are the mean color and covariance of the k-th component of the GMM and w_k are the weights and consider the number of samples closest to the k-th component of the GMM. We use $K = 5$ Gaussians which provides satisfying results in our editing framework.

Prior energies E_I and E_T. We adopt the global link costs from [23] and extend them by geometry information. Besides considering color differences we weight the normal differences between adjacent data samples in a similar way. The intra-frame and inter-frame energies E_I can then be defined in a similar way:

$$E_{I|T} = e^{(-\nabla_c^2/(2\eta_{c,I|T}^2))} + e^{(-\nabla_n^2/(2\eta_{n,I|T}^2))} \quad (6)$$

The gradient ∇ defines the color or normal difference between two nodes u and v which the link connects. The η's represent the intra-frame and inter-frame variance of the color or normal gradients. In our current implementation we do not calculate the variances but empirically set $\eta_c = 0.08$ and $\eta_n = 0.12$.

5 Editing

Editing operations are leveraged using the slicing and selection operators described in the previous sections. Our supported set of editing operations is simple yet becomes very powerful in our framework and with the underlying 4D representation.

Cut & paste. After slicing and selection, the user can employ a clipboard to perform cut or copy operators for selected regions or objects. The data in the clipboard can then be used to paste objects to other hyperslices, other scenes or to clone objects. Compositing of multiple scenes can be done conveniently by first loading all the scenes together at different places in the video hypervolume and then moving their objects around. Objects can be easily removed without leaving holes in the background scene if the acquisition system was able to capture the background behind the object from a suitable viewing angle. Note that compositing in the spatial domain becomes very convenient because our representation explicitly stores the scene geometry.

Transformations. We can apply arbitrary affine transformations to the selection. Transformations are straightforward and intuitive in the case of a hyperslice orthogonal to the t axis. On the other hand, by using other hyperslices we conveniently perform translations in time. To this end, the user simply generates a hyperslice non-orthogonal to the time axis. The translation operator nicely shows the possibilities of a uniform spacetime representation. Other transformation operators include rotation and scaling. The user can rotate objects freely in spacetime, even from the temporal into the spatial domain, creating interesting novel effects like visualization of movement trajectories. Note that in all cases, occlusions are correctly resolved for free by our explicit 3D geometry.

Compositing and shadow mapping. We provide various compositing operators with other media, e.g. images, videos, and virtual objects. They can be inserted into the video hypervolume by conversion to point samples. Alternatively, we allow for insertion of textured meshes during the final rendering phase. To seamlessly blend objects with new backgrounds we adopted a shadow mapping technique [26] to cast shadows of inserted objects into the new background. Again, this is leveraged by the underlying explicit 3D geometry. More realistic compositing can be achieved by adapting the scene's illumination conditions. However, for this purpose time-varying reflectance properties of the scene need to be calculated. This is an interesting challenge for future work.

6 Implementation details

Data structures. Interactive visualization and editing of the video hypervolume requires data structures providing efficient access to the samples. We implemented a two-level approach that relates to the general structure of our editing framework. The first level of the data structure represents the entire four-dimensional video volume. As typical access patterns do not select single points but whole sub-volumes, a regular grid has proven to be very efficient, stored as a spatial hash map for efficient access. Moreover, the grid can be updated very quickly if the user adds, removes or transforms points during the editing session. In our current implementation, the whole video volume still has to fit into the computer's main memory, but this structure is easily extendible to out-of-core data structures that dynamically load the desired grid cells into memory using efficient cashing strategies [12]. The editing itself only takes place in the 3D projection of a selected hyperslice. For efficient rendering, the 3D points are stored as vertex arrays in main memory. Editing operations typically need fast access to single points. Kd-trees are very efficient and widely used for that purpose in traditional point processing frameworks [31]. In our implementation, we build and update a Kd-tree on the fly as soon as a query for a specific point is performed. The Kd-tree can be represented just as a reordering of the vertex arrays. Thus, no additional storage is needed.

Rendering. Slices of the video hypervolume are rendered using EWA splatting [31]. To generate smooth transitions between foreground and background pixels we use a boundary matting technique similar to [30]. It applies an alpha ramp at the boundaries of all objects – edited and non-edited – and renders those splats semi-transparent. Additionally we dynamically compute view-dependent textures like in [4] by back-projecting the images of the color-camera onto the geometry and applying unstructured lumigraph rendering. To achieve correct projection for objects rotated during editing, all changes in orientation are tracked by a transformation attribute for each object.

7 Results and discussion

Our input 3D video footage was generated using a system similar to the one of [24]. It consists of several 3D acquisition bricks that synchronously capture texture and depth maps from their respective viewpoints. For each brick depth information is acquired using a calibrated stereo pair of gray-scale cameras. The stereo matching algorithm is assisted by projectors illuminating the scene with structured light patterns. Instead of alternating the projection with a pattern and its inverse as proposed in the original work, we alternate between a pattern and a black frame to get homogeneously illuminated textures. Four bricks were used to capture a $3 \times 3\text{m}^2$ scene with a convex horizontal viewing range of about 90° (see Fig. 8). We recorded a number of 3D videos and performed editing tasks on the 4D data. The input consists of sequences with a flamenco dancer, an actor juggling a ball and a shot with a plant, a sofa and a sitting person. They were captured at 12 fps and their length was between 80 and 150 frames. Figure 9 displays an example of a reconstructed depth map. The accompanying video shows an interactive editing session and a post-produced 3D video. The latter took approximately one day of editing to complete. For this purpose, our editing system allows for content and viewpoint trajectory scripting.

Figure 10 shows a scene with the flamenco dancer. The dancer was cut out of the original background and inserted into a new one. We used the object selection operator for this purpose. The sequence shows the generation of a "clone" in the same scene and subsequent scaling and transformation to the sofa. Shadow mapping and matting ensures that the dancer still blends in with the new background. Note the shadow in the third image which nicely shows the underlying geometry with the cast shadow of

Fig. 8. Our 3D video studio consisting of four acquisition bricks. Structured light patterns illuminate the scene to support the 3D reconstruction

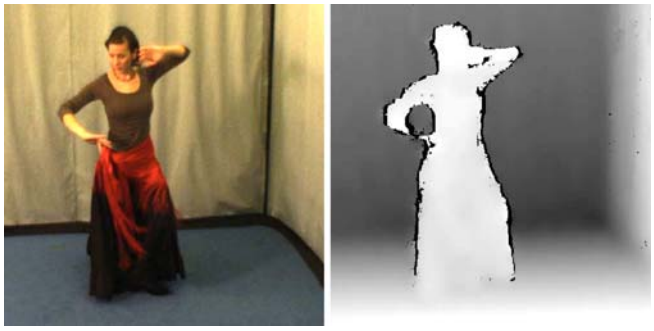

Fig. 9. Example of an acquired color image (left) with corresponding reconstructed depth map (right)

the small dancer onto the sofa. The poster is also inserted onto the wall using the media import feature of our editor. Figure 11 shows an edited 3D video of the juggle sequence cut & paste into the environment with the sofa and the person. The plant shows the limitation of the employed 3D capturing system. Thin structures cannot be handled and the resulting geometry is not captured well. Nevertheless, unstructured lumigraph rendering and matting reduce the resulting artifacts. In this sequence we also replaced the ball with a teapot. The trajectory was captured by cutting out the ball and calculating its center of gravity. We generated spin artificially since we could not capture this from the video footage. Figure 12 combines most of the editing operators in one shot. On the wall we placed a video trailer and applied the Pacific Graphics 2006 logo onto the other wall. Compositing between artificial and real objects is handled very nicely with the boundary matting approach. Some artifacts still remain on the boundaries of

Fig. 10. The Flamenco dancer is inserted into a new environment and cloned. Shadow mapping is applied to seamlessly blend into the scene

Fig. 11. The juggling actor is placed into a new environment. The ball can be replaced by a virtual object following the same trajectory

Fig. 12. The Pacific Graphics 2006 logo and a video trailer are placed onto the walls

objects due to limitations of the acquisition system. The employed stereo matching method has difficulties in accurately reconstructing depth discontinuities. We plan to improve this in the future by applying spatio-temporal segmentation and matting algorithms on the acquired color and depth images.

8 Conclusion

We have demonstrated a system for interactive editing of 3D video footage. It is based on a 4D spatio-temporal representation which allows for unified handling of space and time. Using a three-staged processing loop, we support various editing tasks for post-production of 3D video. For future work we would like to improve our representation by explicitly modeling time coherence [22]. A practical limitation of our current implementation is the large amount of data. We plan to integrate both in-core as well as out-of-core data structures to handle longer 3D video sequences. Furthermore, to speed up the graph-cut object selection, mean-shift pre-segmentation could be employed. Although the presented editing operators allow for the most common editing tasks, others can be envisioned, e.g., altering the motion of actors or re-targeting of motion from one actor to another [6]. In addition, illumination adaptation needs to be solved for application in productive environments. Finally, our system is only as good as the employed 3D video capturing system. We hope that the years to come will provide us with high-quality, commercial depth video scanning systems.

Acknowledgement We would like to thank Vanessa Stadler for her Flamenco performance, Rolf Adelsberger, Patrick Jenni, and Doo Young Kwon for helping with the acquisition, and Christoph Niederberger for directing the video. This work is carried out in the context of the blue-c-II project, funded by ETH grant No. 0-21020-04 as an internal poly-project.

References

1. Bajaj, C.L., Pascucci, V., Rabbiolo, G., Schikorc, D.: Hypervolume visualization: a challenge in simplicity. In: Proc. IEEE/ACM Symposium on Volume Visualization 1998, pp. 95–102 (1998)
2. Bennett, E.P., McMillan, L.: Proscenium: a framework for spatio-temporal video editing. In: Proc. ACM Multimedia 2003, pp. 177–184 (2003)
3. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Transactions on Pattern Analysis and Machine Intelligence **26**(9), 1124–1137 (2004)
4. Buehler, C., Bosse, M., McMillan, L., Gortler, S., Cohen, M.: Unstructured lumigraph rendering. In: Proc. ACM SIGGRAPH 2001, pp. 425–432 (2001)
5. Carranza, J., Theobalt, C., Magnor, M.A., Seidel, H.P.: Free-viewpoint video of human actors. ACM Transactions on Graphics **22**(3), 569–577 (2003)
6. Cheung, G., Baker, S., Hodgins, J., Kanade, T.: Markerless human motion transfer. In: Proc. International Symposium on 3D Data Processing, Visualization and Transmission 2004, pp. 373–378 (2004)
7. Fels, S., Mase, K.: Interactive video cubism. In: Proc. Workshop on New Paradigms in Information Visualization and Manipulation 99, pp. 78–82 (1999)
8. Klein, A.W., Sloan, P.P.J., Finkelstein, A., Cohen, M.F.: Stylized video cubes. In: Proc. ACM SIGGRAPH Symposium on Computer Animation 2002 (2002)
9. Li, Y., Sun, J., Shum, H.Y.: Video object cut and paste. ACM Transactions on Graphics **24**(3), 595–600 (2005)
10. Li, Y., Sun, J., Tang, C.K., Shum, H.Y.: Lazy snapping. ACM Transactions on Graphics **23**(3), 303–308 (2004)
11. Matusik, W., Buehler, C., Raskar, R., Gortler, S.J., McMillan, L.: Image-based visual hulls. In: Proc. ACM SIGGRAPH 2000, pp. 369–374 (2000)
12. Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The grid file: An adaptable, symmetric multikey file structure. ACM Transactions on Database Systems **9**(1), 38–71 (1984)
13. Pasko, A., Adzhiev, V., Schmitt, B., Schlick, C.: Constructive hypervolume modeling. Graphical Models **64**(2) (2002)
14. Pauly, M., Gross, M.: Spectral processing of point-sampled geometry. In: Proc. ACM SIGGRAPH 2001, pp. 379–386 (2001)
15. Pauly, M., Gross, M., Kobbelt, L.: Efficient simplification of point-sampled geometry. In: Proc. IEEE Visualization 2002, pp. 163–170 (2002)
16. Pfister, H., Zwicker, M., van Baar, J., Gross, M.: Surfels: surface elements as rendering primitives. In: Proc. ACM SIGGRAPH 2000, pp. 335–342 (2000)
17. Rander, P., Narayanan, P., Kanade, T.: Virtualized reality: Constructing time-varying virtual worlds from real events. In: Proc. IEEE Visualization 1997, pp. 277–283 (1997)
18. Rother, C., Kolmogorov, V., Blake, A.: "GrabCut" – interactive foreground extraction using iterated graph cuts. ACM Transactions on Graphics **23**(3), 309–314 (2004)
19. Sadlo, F., Weyrich, T., Peikert, R., Gross, M.: A practical structured light acquisition

system for point-based geometry and texture. In: Eurographics Symposium on Point-Based Graphics 2005, pp. 89–98 (2005)
20. Snavely, N., Zitnick, L., Kang, S.B., Cohen, M.: Stylizing 2.5-d video. In: Proc. International Symposium on Non-Photorealistic Animation and Rendering 2006, pp. 63–69 (2006)
21. Theobalt, C., Ahmed, N., de Aguiar, E., Ziegler, G., Lensch, H., Magnor, M.A., Seidel, H.P.: Joint motion and reflectance capture for creating relightable 3D videos. Research Report MPI-I-2005-4-004, Max-Planck-Institut für Informatik (2005)
22. Vedula, S., Baker, S., Rander, P., Collins, R., Kanade, T.: Three-dimensional scene flow. IEEE Transactions on Pattern Analysis and Machine Intelligence 2005 **27**(3), 475–480 (2005)
23. Wang, J., Bhat, P., Colburn, R.A., Agrawala, M., Cohen, M.F.: Interactive video cutout. ACM Transactions on Graphics **24**(3), 585–594 (2005)
24. Waschbüsch, M., Würmlin, S., Cotting, D., Sadlo, F., Gross, M.: Scalable 3D video of dynamic scenes. The Visual Computer **21**(8–10), 629–638 (2005)
25. Weyrich, T., Pauly, M., Keiser, R., Heinzle, S., Scandella, S., Gross, M.: Post-processing of scanned 3D surface data. In: Eurographics Symposium on Point-Based Graphics 2004, pp. 85–94 (2004)
26. Williams, L.: Casting curved shadows on curved surfaces. In: Proc. ACM SIGGRAPH 1978, pp. 270–274 (1978)
27. Woodring, J., Wang, C., Shen, H.W.: High dimensional direct rendering of time-varying volumetric data. In: Proc. IEEE Visualization 2003, pp. 417–424 (2003)
28. Würmlin, S., Lamboray, E., Gross, M.: 3D video fragments: Dynamic point samples for real-time free-viewpoint video. Computers & Graphics **28**(1), 3–14 (2004)
29. Würmlin, S., Lamboray, E., Staadt, O.G., Gross, M.H.: 3D video recorder. In: Proc. Pacific Graphics 2002, pp. 325–334 (2002)
30. Zitnick, C.L., Kang, S.B., Uyttendaele, M., Winder, S., Szeliski, R.: High-quality video view interpolation using a layered representation. ACM Transactions on Graphics **23**(3), 600–608 (2004)
31. Zwicker, M., Pauly, M., Knoll, O., Gross, M.: Pointshop 3D: an interactive system for point-based surface editing. ACM Transactions on Graphics **21**(3), 322–329 (2002)

MICHAEL WASCHBÜSCH currently a Ph.D. candidate in the Computer Graphics Laboratory at ETH Zurich. In 2003, he received his computer science diploma degree from the University of Kaiserslautern, Germany. His research interests include 3D video, 3D reconstruction, point-based rendering and graphics hardware.

DR. STEPHAN WÜRMLIN is currently a postdoctoral researcher in the Computer Graphics Laboratory at ETH Zurich and managing director of the blue-c-II project (http://blue-c-II.ethz.ch). He received a diploma degree in computer science engineering from ETH Zurich in 2000 and a PhD degree in computer graphics from ETH Zurich in 2004. His PhD thesis focused on the design of the 3D video technology for the blue-c collaborative virtual reality system. Part of this has been adopted by MPEG as an extension of the MPEG-4 standard. Dr. Würmlin is member of ACM, ACM SIGGRAPH and ACM SIGCHI. His current research interests include free-viewpoint video, video-based rendering, real-time rendering, virtual reality and multimedia coding. Dr. Würmlin is co-founder and CEO of LiberoVision AG, a company focused on virtual content enhancement technologies for sports broadcasting. He was awarded the "Venture Leaders" young entrepreneur's prize by the Gebert Rüf Foundation and VentureLab in 2006.

DR. MARKUS GROSS is a professor of computer science and director of the Computer Graphics Laboratory of the Swiss Federal Institute of Technology (ETH) in Zurich. He received a master of science in electrical and computer engineering and a PhD in computer graphics and image analysis, both from the University of Saarbrucken, Germany. From 1990 to 1994, Dr. Gross worked for the Computer Graphics Center in Darmstadt, where he established and directed the Visual Computing Group. His research interests include point-based graphics, physics-based modeling, multiresolution analysis, and virtual reality. He has been widely publishing and lecturing on computer graphics and scientific visualization, and he authored the book "Visual Computing", Springer, 1994. Dr. Gross has taught courses at major graphics conferences including ACM SIGGRAPH, IEEE Visualization, and Eurographics. He is the associate editor of the IEEE Computer Graphics and Applications and has served as a member of international program committees of many graphics conferences. Dr. Gross has been a papers co-chair of the IEEE Visualization '99, the Eurographics 2000, and the IEEE Visualization 2002 conferences. He was chair of the papers committee of ACM SIGGRAPH 2005. Dr. Gross is a senior member of IEEE, a member of the IEEE Computer Society, a member of ACM and ACM Siggraph, and a member of the Eurographics Association. Dr. Gross is on the advisory boards of various international research institutes and governmental agencies. Dr Gross is a cofounder of Cyfex AG, Novodex AG and LiberoVision AG.